

CRO

Programmieren eines mobilen Roboters in C++
mit einem Raspberry Pi als Steuerungsplattform

23.9.2019

Wintersemester 2019

Robert Amann



Inhaltsverzeichnis

1	Einleitung.....	3
1.1	Projektbeschreibung.....	3
1.2	Mögliche Erweiterungen.....	3
1.3	Projektabgabe.....	3
1.4	Organisation.....	5
1.5	Literatur	5
2	Windows Konsolenapplikation.....	6
2.1	SimMotor	6
2.2	win_main.cpp (1).....	6
2.3	SimEncoder	7
2.4	win_main.cpp (2)	9
2.5	Shared Files	10
2.6	DifferentialDrive.....	11
2.7	Odometry.....	11
2.8	main_win.cpp (3)	12
2.9	Abstraktion	13
3	Raspberry Pi Konsolenapplikation.....	15
3.1	RPiMotor.....	15
3.2	RPiEncoder.....	16
3.3	Kompass.....	17
3.4	rpi_main.cpp	17
4	FAQs	19
	Anhang.....	20

1 Einleitung

1.1 Projektbeschreibung

Programmieren Sie die Steuerung eines mobilen Roboters.

- Der Roboter nutzt zwei Motoren (Differentialantrieb)
- Der Roboter soll sich in einem Vierecks bewegen.
- Der Roboter nutzt zwei Encoder für Odometrie.
- Der Roboter nutzt einen Magnetometer zur Orientierungsbestimmung.
- Wenn der Magnetometer angeschlossen ist, wird dieser zur Richtungsbestimmung genutzt, sonst die zwei Encoder.
- Die Steuerung erfolgt durch einen Raspberry Pi, programmiert in C++.

Es sollen zwei Programme erstellt werden: eine Windows Konsolenapplikation und eine Raspberry Pi Konsolenapplikation. Beide Programme sollen die abstrakten Basisklassen Motor und Encoder und die Klassen Odometrie und DifferentialDrive verwenden. Die Windows Konsolenapplikation soll einen Motor und einen Encoder simulieren. Die Raspberry Pi Konsolenapplikation soll den mobilen Roboter steuern.

1.2 Mögliche Erweiterungen

- Update der Robotersoftware auf Version 2.0
 - Klasse RPiKeys zum Auslesen der Tasten am Explorer Hat Pro (zum Beispiel um den Roboter zu Starten/Stoppen), Nutzung der LEDs
 - Abfahren eines komplexeren (programmierbaren) Pfades
 - Handlungsplanung (ausweichen, umdrehen, etc.) in Abhängigkeit von Sensorsignalen (können auch Taster am Explorer hat sein)
- Bildverarbeitung mit OpenCV
 - Beobachtung des Roboters mit Kamera
 - Bildverarbeitung mit OpenCV (in der Programmiersprache C++ auf PC)
 - Senden von Steuerbefehlen von PC an Roboter über TCP/IP zur Kollisionsvermeidung
- Integration eines Lidar, Fahren entlang der Wände (Achtung: Beschaffungszeit Lidar 1-3 Wochen!)
- Fernsteuerung des mobilen Roboters mittels PC (in der Programmiersprache C++ auf PC), frei programmierbare Positionen (in kartesischen Koordinaten) können angefahren werden (Pfadplanung)
- Nutzung von Entwurfsmustern, zum Beispiel *Singleton* und *State Design Pattern*

1.3 Projektabgabe

Abgabe im ILIAS: Geben Sie Ihre Visual Studio Solution als ZIP-Datei verpackt im ILIAS ab. Dokumentieren Sie Änderungen und Erweiterungen in einer eigenen Textdatei.

1.4 Organisation

In einem ersten Schritt wird eine Windows Konsolenapplikation zur Ansteuerung eines Zweiradfahrzeuges mit Differentialantrieb entwickelt.

In einem zweiten Schritt wird der Raspberry Pi 3 genutzt, um einen echten Roboter anzusteuern.

1.5 Literatur

Unterstützende Literatur zur C++ Programmierung

Louis, Dirk (2014): C++: Das komplette Starterkit für den einfachen Einstieg in die Programmierung. München: Carl Hanser Verlag GmbH & Co. KG. Online im Internet: DOI: [10.3139/9783446441095](https://doi.org/10.3139/9783446441095)

Kaiser, Richard (2018): C++ mit Visual Studio 2017. Berlin, Heidelberg: Springer Berlin Heidelberg. Online im Internet: DOI: [10.1007/978-3-662-49793-7](https://doi.org/10.1007/978-3-662-49793-7)

Weiterführende Literatur zum Thema mobile Roboter

Hertzberg, Joachim; Lingemann, Kai; Nüchter, Andreas (2012): Mobile Roboter: eine Einführung aus Sicht der Informatik. Berlin: Springer Vieweg.

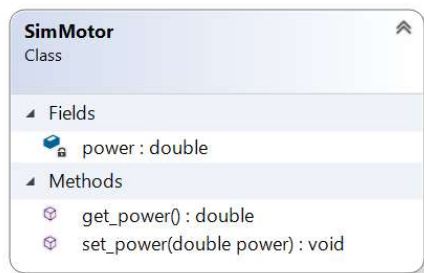
Dudek, Gregory; Jenkin, Michael (2010): Computational principles of mobile robotics. 2. Cambridge: Cambridge University Press.

2 Windows Konsolenapplikation

In diesem Kapitel werden Klassen zur Ansteuerung eines Zweiradfahrzeuges mit Differentialantrieb (differential wheeled robot) entwickelt und in einer Windows Konsolenapplikation getestet.

2.1 SimMotor

- Erstellen Sie eine Klasse `SimMotor` und implementieren Sie die Methoden `set_power` und `get_power`. Die Leistung eines Motors kann den Wert 0-100% annehmen.



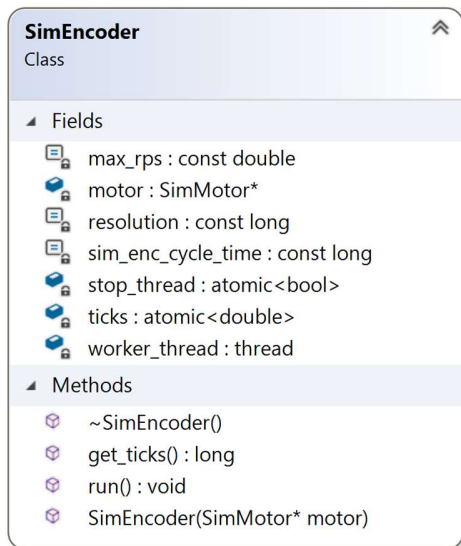
- Erstellen Sie in der Datei `CRO.cpp` im `main()` ein Objekt der Klasse `SimMotor`, testen Sie die Methoden `set_power` und `set_power` und löschen Sie das Objekt wieder.

2.2 win_main.cpp (1)

```
int main()
{
    SimMotor* m_left = new SimMotor();
    // Test der Methoden set_power und get_power
    delete m_left;
}
```

2.3 SimEncoder

Um einen Encoder in der Windows Konsolenapplikation zu simulieren, wird eine Klasse programmiert, die einen eigenen Thread nutzt, um einen Winkelgeber (Encoder) zu simulieren. In der Methode `run()`, die in diesem Thread nebenläufig zum Hauptprogramm ausgeführt wird, wird der Zählerstand `ticks` entsprechend dem Zustand des Motors: `motor->get_power()` erhöht.



```
// Klasse SimEncoder
//
// LV:      Embedded Systems 1
// Datum:   Wintersemester 2019
// Autor:   Robert Amann
//
// SimEncoder
// - simuliert einen Drehgeber.
// - hat einen zugehörigen Motor und einen Zählerstand.
// - aktualisiert abhängig von der eingestellten Leistung
//   des Motors seinen Zählerstand.
// - nutzt einen Thread, um den Zählerstand im 100ms-Takt zu aktualisieren.

#pragma once
#include <thread>
#include <atomic>
#include <chrono>
#include "SimMotor.h"
```

```

class SimEncoder
{
private:
    std::atomic<double> ticks;           // aktueller Zählerstand
    SimMotor* motor;                   // Motor, der den Drehgeber bewegt
    const long sim_enc_cycle_time = 100; // Zykluszeit der Simulation in Millisekunden
    const double max_rps = 1;          // Maximale Drehzahl des Motors
                                        // in Umdrehungen pro Sekunde
    const long resolution = 20;         // Auflösung des Drehgebers
    std::thread worker_thread;
    std::atomic<bool> stop_thread;      // wenn stop_thread == true,
                                        // soll der Thread enden

public:
    // Parameter motor: Motor, der den Drehgeber bewegt
    SimEncoder(SimMotor* motor) {
        stop_thread = false;
        ticks = 0;
        this->motor = motor;
        worker_thread = std::thread(&SimEncoder::run, this); // start thread
    }

    // Der Destruktor setzt stop_thread auf true und wartet
    // bis die run()-Methode des Thread beendet ist.
    ~SimEncoder() {
        stop_thread = true;
        worker_thread.join();
    }

    // run() - Methode des Threads
    // Thread läuft, bis stop_thread (vom Destruktor) auf false gesetzt wird.
    void run() {
        while (!stop_thread) {
            std::this_thread::sleep_for(
                std::chrono::milliseconds(sim_enc_cycle_time));
            ticks += ...;
        }
    }

    // Rückgabewert: aktueller Zählerstand
    long get_ticks() {
        return (long)ticks; // Typwandlung des Zählerstandes (ganze Zahl)
    }
};

```

Für die fehlende Berechnung der ticks benötigen Sie `motor->get_power()`, `max_rps`, `resolution` und `sim_enc_cycle_time/1000`.

2.4 win_main.cpp (2)

Implementieren Sie eine Testfunktion für den Encoder.

```
#include <iostream>
#include <chrono>
#include <thread>
#include "SimMotor.h"
#include "SimEncoder.h"
#include "DifferentialDrive.h"
#include "Odometry.h"

using namespace std;

// Testfunktion für SimEncoder
// Input: Motor und Encoder

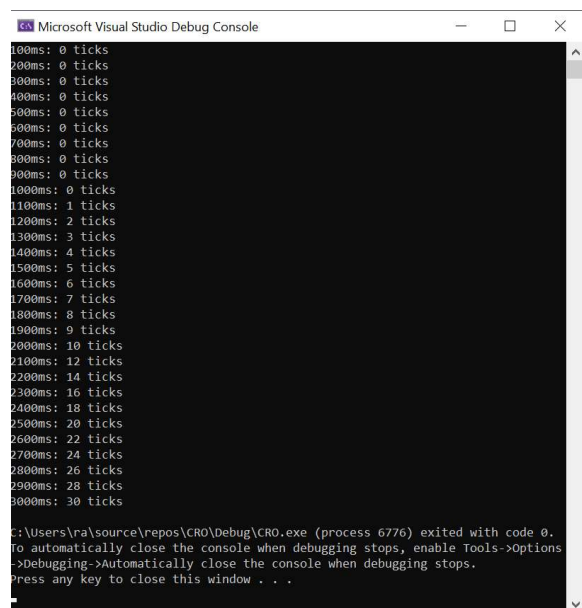
void test_1(SimMotor* motor, SimEncoder* encoder)
{
    // Zähler von 1 bis 30
    //   this_thread::sleep_for 100 Millisekunden
    //   Konsolenausgabe der verstrichenen Zeit in Millisekunden und des Zählerstandes des Encoders
    //   Wenn der Zählerstand == 10, dann soll die Power des Motors auf 50 gesetzt werden
    //   Wenn der Zählerstand == 20, dann soll die Power des Motors auf 100 gesetzt werden
}

int main()
{
    SimMotor* motor_left = ...
    SimEncoder* encoder_left = ...

    test_1(motor_left, encoder_left);

    delete encoder_left;
    delete motor_left;
    return 0;
}
```

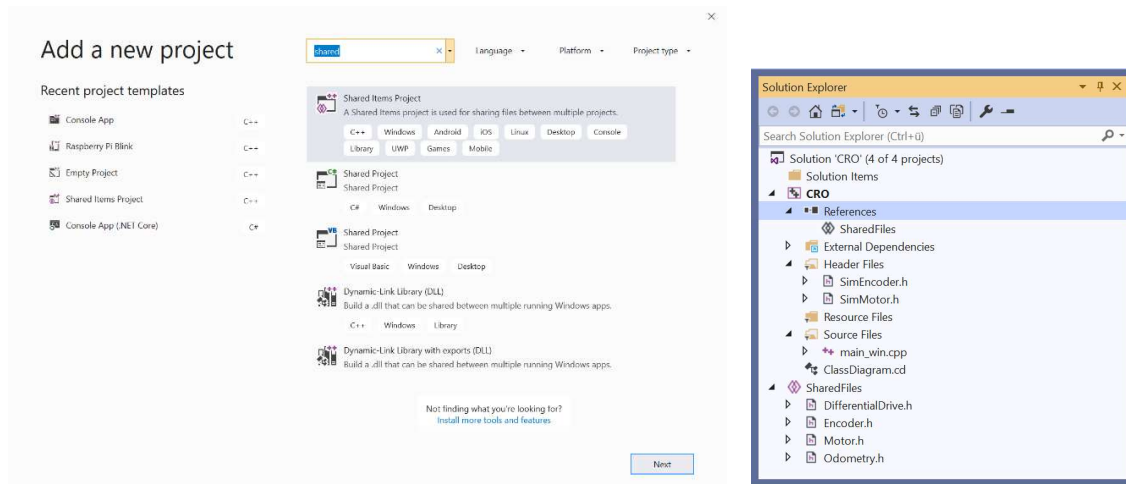
Die Konsolenausgabe soll dann etwa wie folgt aussehen:



```
Microsoft Visual Studio Debug Console
100ms: 0 ticks
200ms: 0 ticks
300ms: 0 ticks
400ms: 0 ticks
500ms: 0 ticks
600ms: 0 ticks
700ms: 0 ticks
800ms: 0 ticks
900ms: 0 ticks
1000ms: 0 ticks
1100ms: 1 ticks
1200ms: 2 ticks
1300ms: 3 ticks
1400ms: 4 ticks
1500ms: 5 ticks
1600ms: 6 ticks
1700ms: 7 ticks
1800ms: 8 ticks
1900ms: 9 ticks
2000ms: 10 ticks
2100ms: 12 ticks
2200ms: 14 ticks
2300ms: 16 ticks
2400ms: 18 ticks
2500ms: 20 ticks
2600ms: 22 ticks
2700ms: 24 ticks
2800ms: 26 ticks
2900ms: 28 ticks
3000ms: 30 ticks
c:\Users\ra\source\repos\CRO\Debug\CRO.exe (process 6776) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options
->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

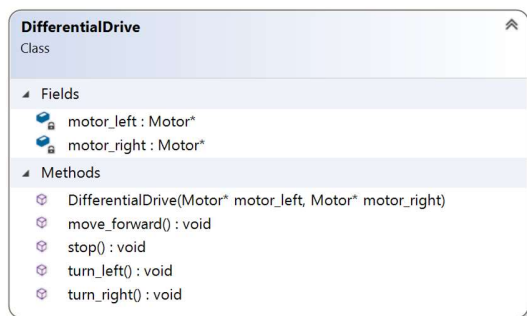
2.5 Shared Files

Die vier Klassen DifferentialDrive, Odometry, Motor und Encoder sollen von zwei Projekten (Windows, Raspberry Pi) genutzt werden. Dazu kann ein „Shared Items Project“ genutzt werden:



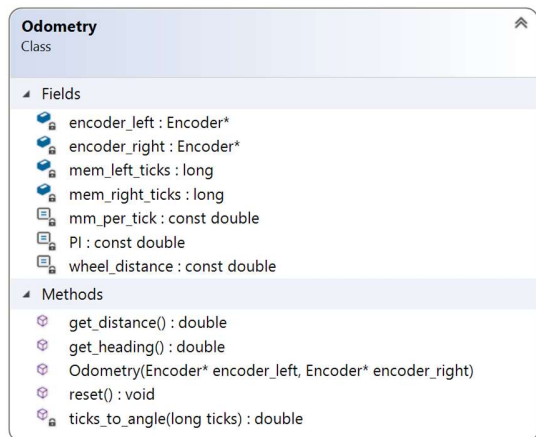
2.6 DifferentialDrive

Ein DifferentialDrive hat einen linken und einen rechten Motor. Es ist verantwortlich für das Setzen der Geschwindigkeit der beiden Motoren. Zum Beispiel wenn `move_forward()` aufgerufen wird, soll der Roboter sich nach vorwärts bewegen. `turn_left()` und `turn_right()` führen zu einer Rechts- respektive Linksdrehung des Roboters. Die Bewegung wird solange fortgeführt, bis `stop()` aufgerufen wird.



2.7 Odometry

Die Odometrie kann genutzt werden, um den zurückgelegten Weg und die aktuelle Orientierung des Roboters zu bestimmen.



Code Snippets:

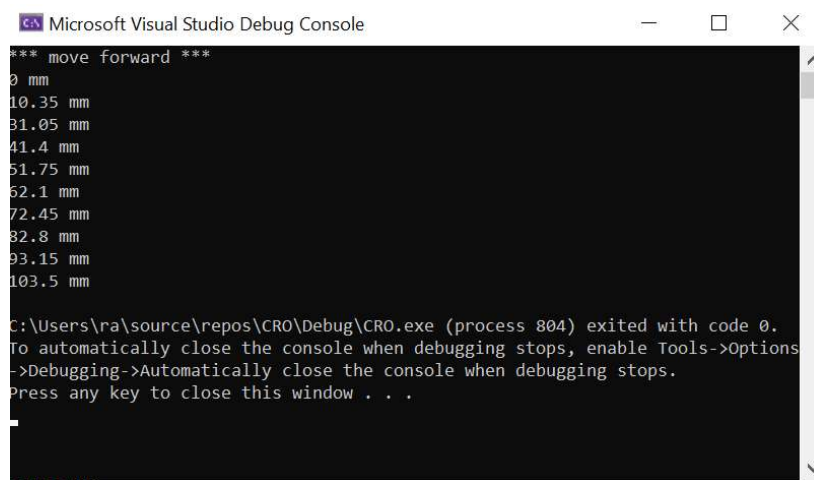
```
const double mm_per_tick = 207.0 / 20.0; // Radumfang / Auflösung in mm
const double wheel_distance = 131;       // Radabstand in mm
const double PI = 3.141592654;
```

2.8 main_win.cpp (3)

```
void test_2(DifferentialDrive* drive, Odometry* odometry)
{
    odometry->reset();
    drive->move_forward();

    // Solange die gefahrene Distanz < 100mm ist:
    //   this_thread::sleep_for 100 Millisekunden
    //   Konsolenausgabe des Zählerstandes des Encoders
}
```

Die Konsolenausgabe soll dann etwa wie folgt aussehen:



```
Microsoft Visual Studio Debug Console

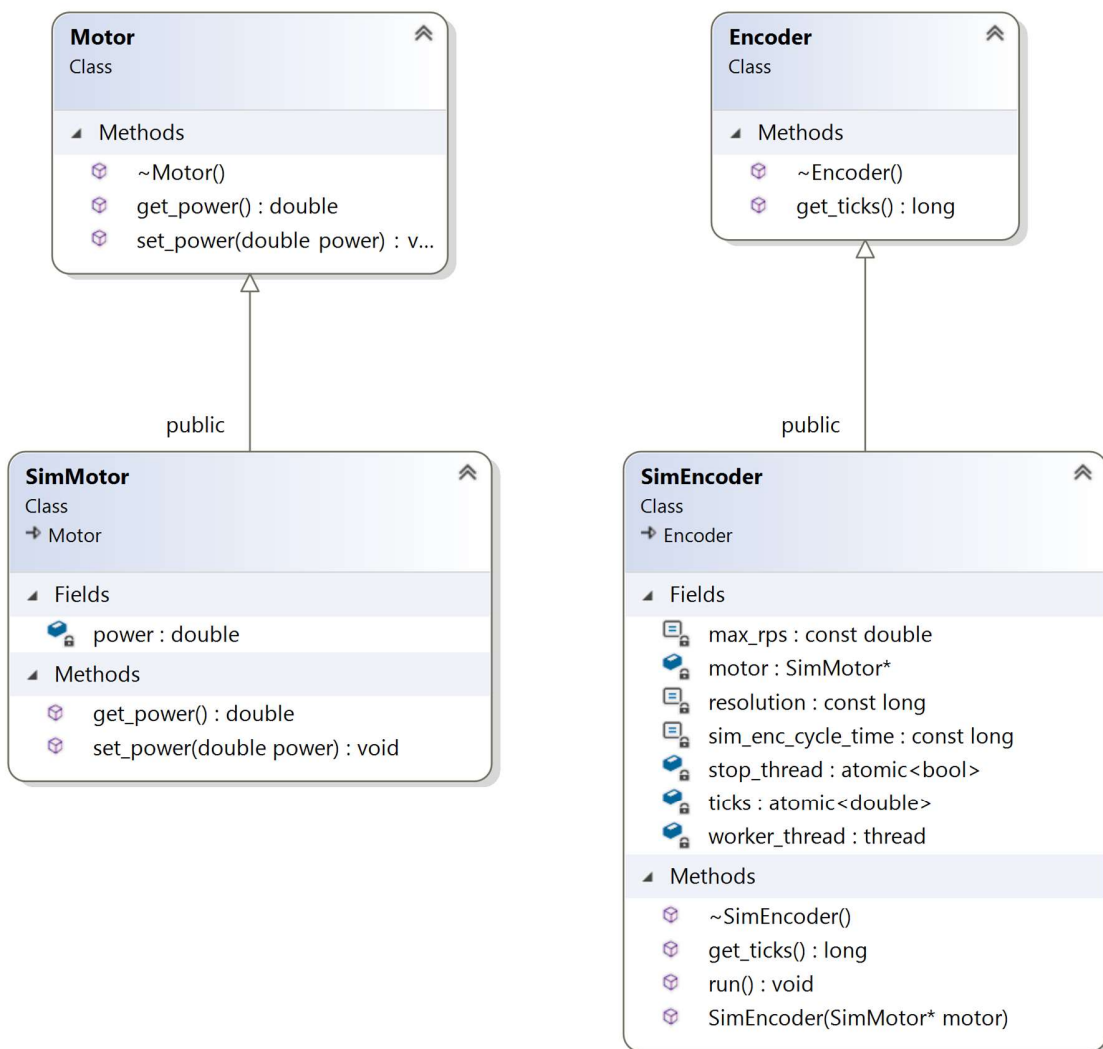
*** move forward ***
0 mm
10.35 mm
31.05 mm
41.4 mm
51.75 mm
62.1 mm
72.45 mm
82.8 mm
93.15 mm
103.5 mm

C:\Users\ra\source\repos\CRO\Debug\CRO.exe (process 804) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options
->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

2.9 Abstraktion

Deklariieren Sie zwei rein abstrakte Klassen Motor und Encoder und machen Sie SimMotor und SimEncoder zu Kind-Klassen von Motor und Encoder.

```
class Motor
{
public:
    virtual void set_power(double power) = 0;
    virtual double get_power() = 0;
    virtual ~Motor() {};
};
```



Nutzen Sie in der Deklaration von DifferentialDrive und Odometry ausschließlich Motor und Encoder (nicht SimMotor und SimEncoder). So können im nächsten Kapitel DifferentialDrive und Odometry ohne Änderung mit den Motoren und Encoder (RPiMotor und RPiEncoder) vom Raspberry Pi verwendet werden.

DifferentialDrive
Class

Fields

motor_left : Motor*

motor_right : Motor*

Methods

DifferentialDrive(Motor* motor_left, Motor* motor_right)

move_forward() : void

stop() : void

turn_left() : void

turn_right() : void

Odometry
Class

Fields

encoder_left : Encoder*

encoder_right : Encoder*

mem_left_ticks : long

mem_right_ticks : long

mm_per_tick : const double

PI : const double

wheel_distance : const double

Methods

get_distance() : double

get_heading() : double

Odometry(Encoder* encoder_left, Encoder* encoder_right)

reset() : void

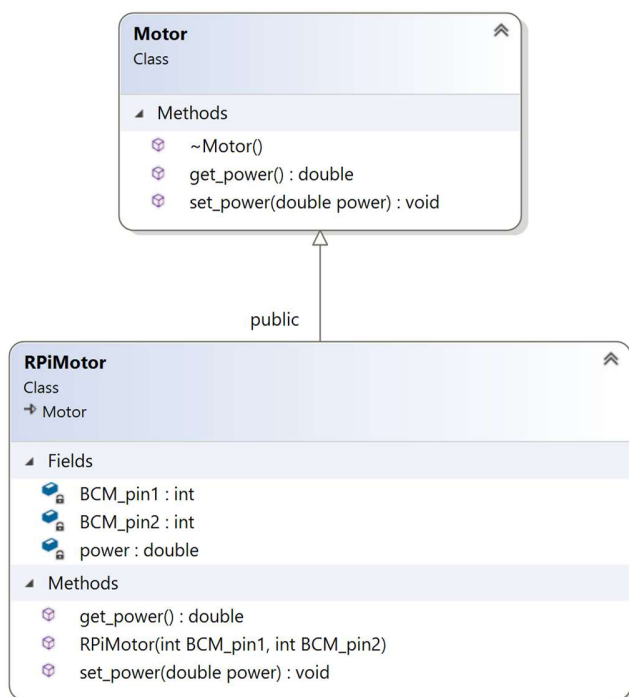
ticks_to_angle(long ticks) : double

3 Raspberry Pi Konsolenapplikation

Weblinks zum Explorer HAT Pro:

- [Explorer HAT Pro Pinout](#)
- [WiringPI Pinout](#)
- [Hardwarebeschreibung der Komponenten des Explorer HAT Pro](#)
- [Shop](#)

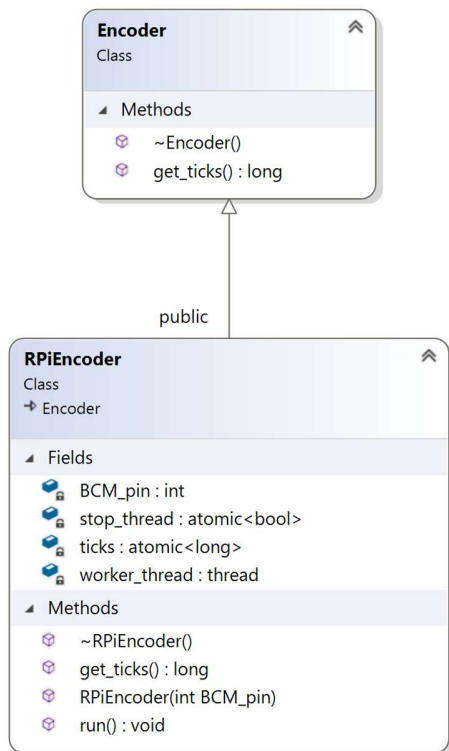
3.1 RPiMotor



Code Snippets:

```
digitalWrite(BCM_pin1, 0);
softPwmWrite(BCM_pin2, (int)power);
```

3.2 RPiEncoder



Code Snippets:

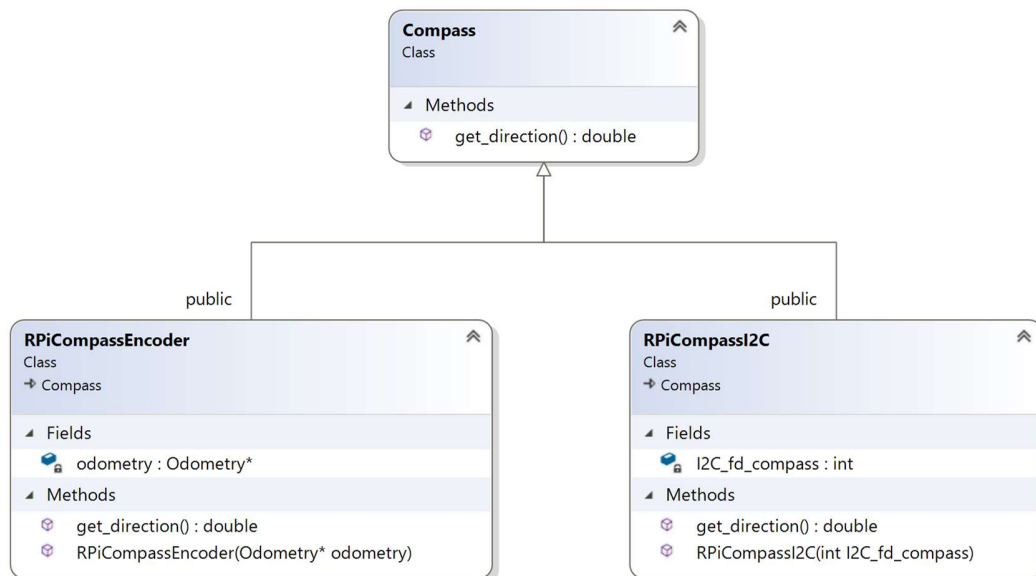
Konstruktor des RPiEncoder mit BCM_pin anstelle Motor*

```
atomic<long> ticks; (statt atomic<double> ticks);
```

```
while (!stop_thread) {
    x = digitalRead(BCM_pin);
    // wenn x von 0 auf 1 wechselt (steigende Flanke) , dann ticks++;
    this_thread::sleep_for(chrono::milliseconds(1)); // works also: delay(1);
}
```


3.3 Kompass

- Nutzen Sie den elektronischen Kompass [CMPS14](#). Achtung: **Der elektronische Kompass funktioniert nicht in der Nähe von Metall**



3.4 rpi_main.cpp

```
const int LEDs_BCM[] = { 4, 17, 27, 5 };
const int Motor_BCM[] = { 19, 20, 21, 26 };
const int Encoder_BCM[] = { 22, 23 };

void setup_explorer_hat() {
    wiringPiSetupGpio();
    for (int i = 0; i < 4; i++) {
        pinMode(LEDs_BCM[i], OUTPUT);
        softPwmCreate(Motor_BCM[i], 0, 100);
    }
    pinMode(Encoder_BCM[0], INPUT);
    pinMode(Encoder_BCM[1], INPUT);
}

void move_forward(DifferentialDrive* drive, Odometry* odometry, double distance) {
    // Odometrie reset, Drive "move forward"
    // solange die Distanz nicht erreicht wurde:
    //   this_thread::sleep für 1 Millisekunde
    // Drive "stop"
}

void turn(DifferentialDrive* drive, Compass* compass, double direction) {
    // solange die Ausrichtung des Roboters mehr als 3 Grad von „direction“ abweicht:
    //   Drive "turn_right" bzw. "turn_left"
}
```

```

int main(void)
{
    /* create objects */

    /* create compass */
    Compass* compass;
    int I2C_fd_compass = wiringPiI2CSetup(0x60);
    if (wiringPiI2CReadReg8(I2C_fd_compass, 1) != -1) { // I2C compass available
        compass = new RPiCompassI2C(I2C_fd_compass);
        cout << "I2C compass" << endl;
    }
    else {
        compass = new RPiCompassEncoder(odometry);
        cout << "Encoder compass" << endl;
    }

    /* main loop */

    // Endlos-Schleife:
    //   Ausrichten des Roboters auf 0° (mittels Funktion turn)
    //   500mm vorwärts fahren (mittels Funktion move_forward)
    //   Ausrichten des Roboters auf 180° (mittels Funktion turn)
    //   500mm rückwärts fahren (mittels Funktion move_forward)

}

```

4 FAQs

WLAN hotspot Windows 10

Wenn Sie Ihr Notebook mit Windows 10 als WLAN Hotspot für den Raspberry Pi nutzen, dann geben Sie bei der Konfiguration des WI-FI Adapters im Windows 10 darauf acht, dass der Adapter für die Verbindung ins Heimnetzwerk (oder FHV-Netzwerk) nur das 2.4 GHz Band nutzt; dann ist das 5GB Band frei für die Verbindung mit dem Raspberry Pi.

Check, ob Kompass angeschlossen

Zur Überprüfung ob der Kompass am Raspberry Pi richtig angeschlossen ist, können Sie folgenden Aufruf nutzen:

```
2cdetect -y 1
```

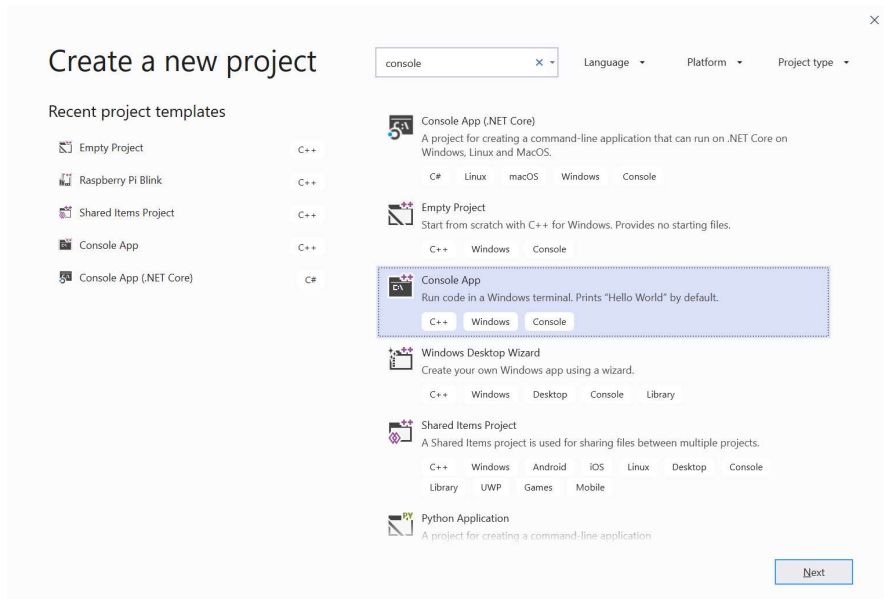
Anhang

Anhang A	Visual Studio Setup.....	21
A-1	Console App für Windows.....	21
A-2	Visual Studio Setup für Raspberry Pi Linux Development with C++.....	21
A-3	Console App für Raspberry Pi	22
Anhang B	Raspberry Pi Setup.....	24
B-1	Raspian installieren	24
B-2	Konfiguration.....	24
B-3	Statische IP-Adresse.....	24
B-4	Update	24
B-5	Explorer Hat Pro.....	25
Anhang C	Raspberry Pi Hardware	26

Anhang A Visual Studio Setup

A-1 Console App für Windows

- Erstellen Sie mit Visual Studio 2019 eine neue Solution. Eine Solution kann mehrere Projekte verwalten.
- Erstellen Sie ein neues Projekt (Console App) mit dem Namen CRO¹ oder einem Namen Ihrer Wahl.



A-2 Visual Studio Setup für Raspberry Pi Linux Development with C++

<https://docs.microsoft.com/en-us/cpp/linux/?view=vs-2019>

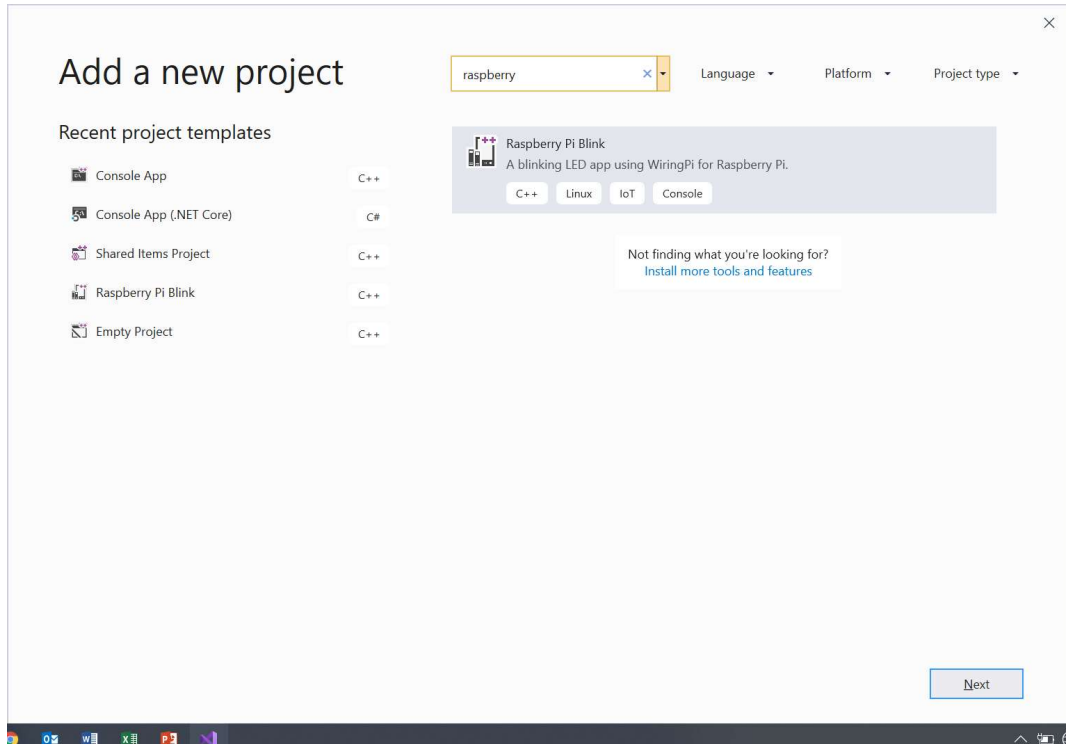
Einstellungen bei Connect to Remote System:

Host Name (IP address)	192.168.0.2
Port	22
User Name	pi
Password	mensa10

¹ CRO steht für C++ robot

A-3 Console App für Raspberry Pi

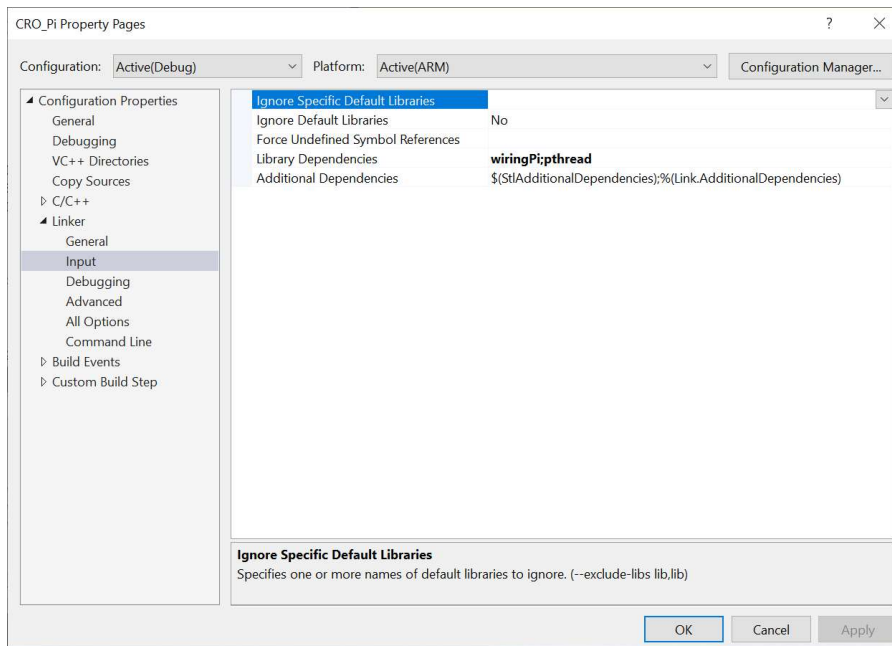
- Erstellen Sie mit Visual Studio 2019 ein neues Projekt (Console App) mit dem Namen CRO_Pi oder einem Namen Ihrer Wahl.



wiringPiSetupGpio()

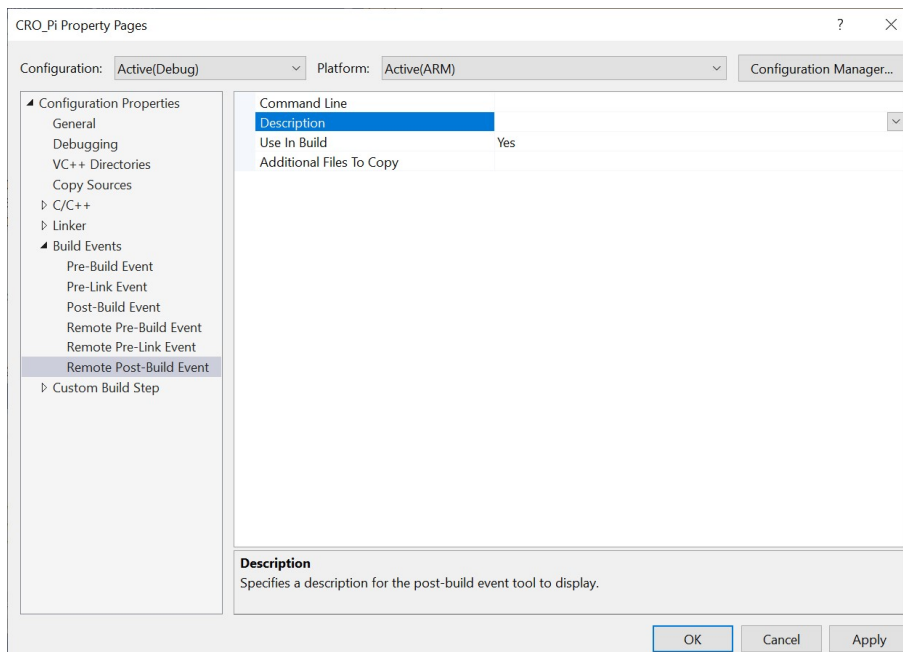
wiringPiSetupGpio(); anstelle von wiringPiSetupSys(); wegen softPWM

Linking pthread



Remote Post-Build Event

Remove: Remote Post-Build Event: empty Command Line



Anhang B Raspberry Pi Setup

Diese Einstellungen sind auf dem Raspberry Pi, der Ihnen zur Verfügung steht, schon erfolgt.

B-1 Raspian installieren

<https://www.raspberrypi.org/downloads/raspbian/>

installierte Version: **Raspbian Buster Full**

B-2 Konfiguration

- Localisation – Austria
- Passwort ändern: mensa10
- Connect to WIFI (Mobiltelefon)

```
$ sudo raspi-config
```

- Boot options: Console, do not wait for network, no splash screen
- Interfacing options: SSH aktivieren, I2C aktivieren

B-3 Statische IP-Adresse

<https://www.elektronik-kompendium.de/sites/raspberry-pi/1912151.htm>, Variante 2:

```
$ sudo nano /etc/dhcpd.conf
```

```
interface eth0
static ip_address=192.168.1.2/24
static routers=192.168.1.1
static domain_name_servers=192.168.1.1 (für ein schnelleres Login!)
```

Beim PC den USB-Ethernet Koppler verwenden und folgende TCP/IPv4-Konfiguration vornehmen:

IP address: 192.168.1.3

Subnet mask: 255.255.255.0

B-4 Update

```
sudo apt update
sudo apt upgrade
```


B-5 Explorer Hat Pro

[https://learn.pimoroni.com/tutorial/sandyj/explorer-hat-pro-pin-entry-system:](https://learn.pimoroni.com/tutorial/sandyj/explorer-hat-pro-pin-entry-system)

```
curl https://get.pimoroni.com/i2c | sudo bash
sudo apt-get install python-smbus
sudo apt-get install python-pip
sudo pip install explorerhat
```

Anhang C Raspberry Pi Hardware

Explorer Hat Technical Reference (on board hardware)

<https://github.com/pimoroni/explorer-hat/blob/master/documentation/Technical-reference.md>

